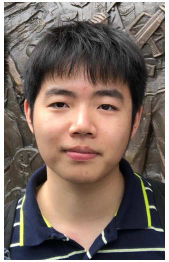
UMass Amherst | College of Information & Computer Sciences

NIM: Generative Neural Networks for Modeling and Generation of Simulation Inputs


Emily A. Herbert


Wang Cen


Peter J. Haas

Stochastic discrete-event simulation

- Improves design and operation of complex engineered systems
- Simulation tools can help specify the simulation model structure
- Modeling simulation inputs remains as a challenging task
  - particularly for non-experts
- **Our goal is to facilitate this process via automation**

Input modeling is challenging

- Data for fitting distributions traditionally is expensive and painful to collect
- So, a modeler typically imposes strong simplifying assumptions
- But, these assumptions can pose their own challenges

Input modeling is challenging

Assume the interarrival times to a system are independent and identically distributed (i.i.d.) → Sacrifice fidelity when capturing complex distribution features (e.g. multimodality)

Input modeling is challenging

Assume the interarrival times to a system are independent and identically distributed (i.i.d.) → Fit an empirical distribution

↓

The data values that can be produced in a simulation run are strictly limited to those in the available data

Input modeling is challenging

- Sometimes a modeler cannot make such simplifying assumptions
- So, they might try different strategies

Input modeling is challenging

The interarrival sequence is not well modeled as a sequence of i.i.d. random variables → Choose between many possible models for autocorrelated, possibly non-stationary interarrival sequences

time-series models

ARIMA

GARCH

SETAR

direct point process models of arrival times (Cox and Isham 1980)

Lots of choices and no software guidance

Input modeling is challenging

The interarrival sequence is not well modeled as a sequence of i.i.d. random variables

Use input traces for general stochastic processes

Introduces privacy issues when deployed

The data values that can be produced in a simulation run are strictly limited to those in the available data

Input modeling is challenging

- But… data is becoming ubiquitous
- And… we observe that neural networks are a powerful tool for learning complex patterns from data in data-rich environments
- **Neural networks are promising for:**
  - **automating the tasks of learning simulation input distributions**
  - **generating samples from these distributions during simulation runs**

Neural Input Modeling (NIM)

- Framework for automated modeling and generation of simulation input distributions
- Uses generative neural networks (GNNs)
- **Learns a complex statistical distribution without overfitting**
- **Provides a means of sampling from the distribution**

Neural Input Modeling (NIM)

- Seeks to resolve...

The data values that can be produced in a simulation run are strictly limited to those in the available data

Sacrifice fidelity when capturing complex distribution features

Introduces privacy issues when deployed

Lots of choices and no software guidance

NIM-VL

- NIM prototype!
- Captures complex stochastic input processes and simulates them
- Neural network of the form:

Variational Autoencoder + LSTM layers

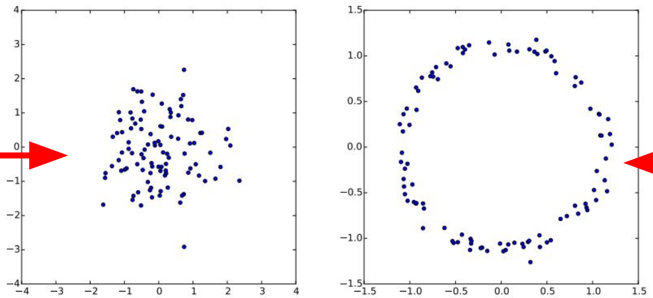
Generative Neural Networks (GNNs)

- Learn features from the training data
- Generate new synthetic data using these learned features
- Generating synthetic faces, music, and sentences
- **Learn simulation input distributions and generate synthetic stochastic processes (e.g. an interarrival sequence)**

Variational Autoencoders (VAEs)

- Type of GNN
- Learn a deterministic function from the training data that allows samples from a known distribution to be transformed to samples of the target, unknown distribution

samples from a gaussian distribution



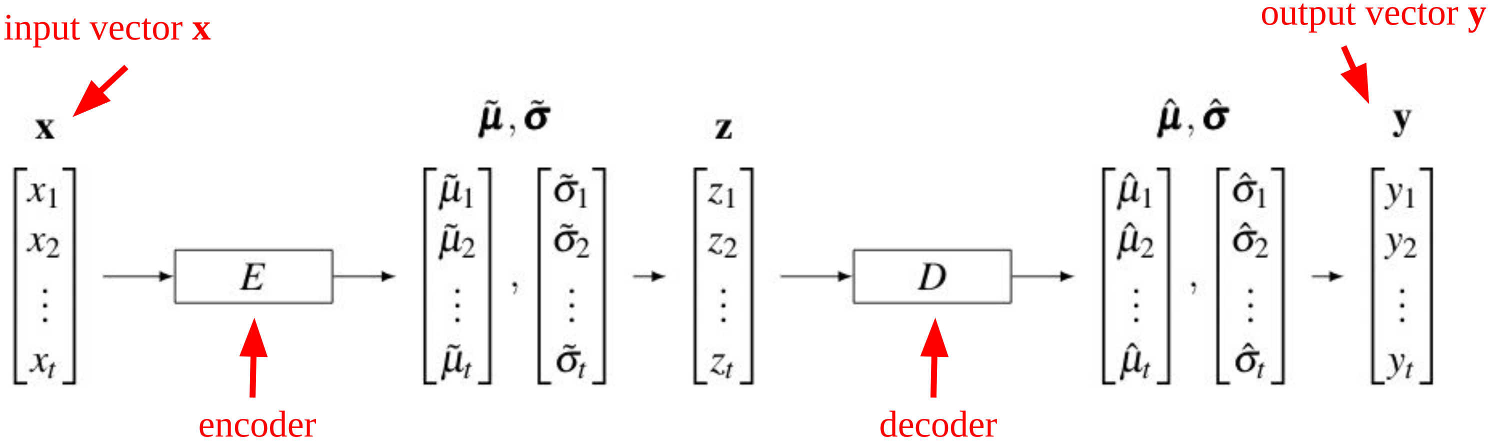
the same samples mapped through $g(z) = z/10 + z/||z||$

VAE learns function g

(Doersch '16)

Variational Autoencoders (VAEs)

- Uses a pair of networks - an encoder E and a decoder D



$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i$$

$$\xi_i \sim N(0,1)$$

$$y_i = \hat{\sigma}_i w_i + \hat{\mu}_i$$

$$w_i \sim N(0,1)$$

"reparameterization"

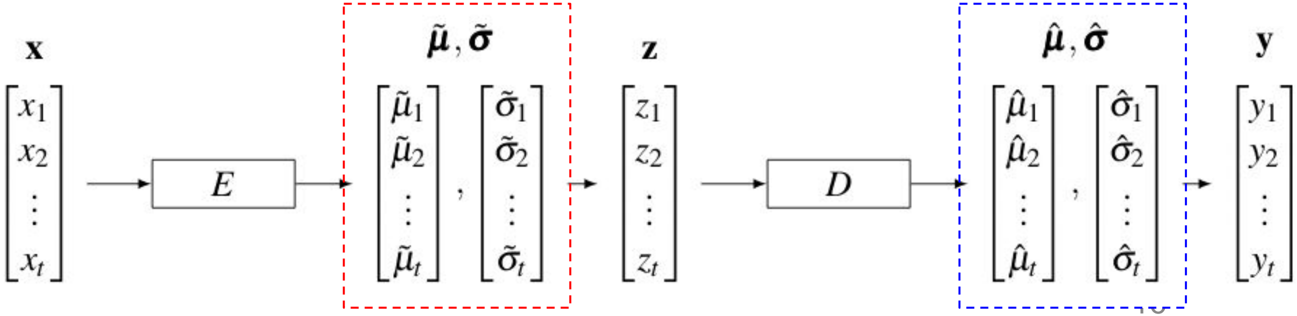
Variational Autoencoders (VAEs)

"Designed with generation in mind"

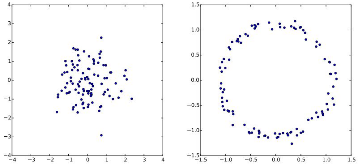
- Goals:

  (i) Given i.i.d N(0,1) random variables **z**, the decoder will produce $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}$ such that **y** will be distributed as a sample from the target distribution

  (ii) The encoder will produce $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}$ such that **z** will be distributed from N(0,1)

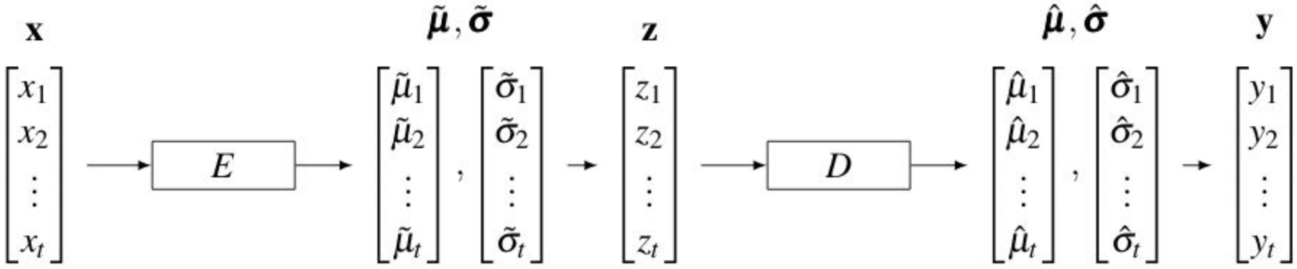


Variational Autoencoders (VAEs)



- During generation:
  - The decoder transforms stochastic **z** from N(0,1) such that **y** will be distributed as a sample from the target distribution

- **Generates data that will be distributed as a sample from the target distribution**



Variational Autoencoders (VAEs)

- During training:
  - Assume *t* arrivals in a day. Observe: $\mathbf{x} = (x_1, x_2, \ldots, x_t)$
  - Learns to stochastically encode **x** into **z** (i.i.d. N(0,1))
  - Learns to decode **z** (i.i.d. N(0,1)) into **y** (reconstructing **x**)
  - Loss: $L(\mathbf{x}, E, D) = D_{KL}[Q(\mathbf{z}|\mathbf{x}, E)||N(\mathbf{0}, \mathbf{I})] - \mathbb{E}_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x},D)}[\log P(\mathbf{x}|\mathbf{z}, D)]$

$$\overset{\mathbf{x}}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \end{bmatrix}} \longrightarrow \boxed{E} \longrightarrow \overset{\tilde{\boldsymbol{\mu}},\, \tilde{\boldsymbol{\sigma}}}{\begin{bmatrix} \tilde{\mu}_1 \\ \tilde{\mu}_2 \\ \vdots \\ \tilde{\mu}_t \end{bmatrix}, \begin{bmatrix} \tilde{\sigma}_1 \\ \tilde{\sigma}_2 \\ \vdots \\ \tilde{\sigma}_t \end{bmatrix}} \rightarrow \overset{\mathbf{z}}{\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_t \end{bmatrix}} \longrightarrow \boxed{D} \longrightarrow \overset{\hat{\boldsymbol{\mu}},\, \hat{\boldsymbol{\sigma}}}{\begin{bmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \\ \vdots \\ \hat{\mu}_t \end{bmatrix}, \begin{bmatrix} \hat{\sigma}_1 \\ \hat{\sigma}_2 \\ \vdots \\ \hat{\sigma}_t \end{bmatrix}} \rightarrow \overset{\mathbf{y}}{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{bmatrix}}$$

Variational Autoencoders (VAEs)

minimize KL-divergence between **z** values and N(0,1) (regularization term)

maximize Log-Likelihood between jointly distributed normal samples from the decoder (reconstruction term)

$$L(\mathbf{x},E,D) = D_{KL}[Q(\mathbf{z}|\mathbf{x},E)||N(\mathbf{0},\mathbf{I})] - \mathbb{E}_{\mathbf{z}\sim Q(\mathbf{z}|\mathbf{x},D)}[\log P(\mathbf{x}|\mathbf{z},D)]$$

$$\begin{array}{ccccccccc} \mathbf{x} & & \tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}} & & \mathbf{z} & & \hat{\boldsymbol{\mu}},\hat{\boldsymbol{\sigma}} & & \mathbf{y} \\ \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_t \end{bmatrix} & \longrightarrow \boxed{E} \longrightarrow & \begin{bmatrix} \tilde{\mu}_1 \\ \tilde{\mu}_2 \\ \vdots \\ \tilde{\mu}_t \end{bmatrix}, \begin{bmatrix} \tilde{\sigma}_1 \\ \tilde{\sigma}_2 \\ \vdots \\ \tilde{\sigma}_t \end{bmatrix} & \rightarrow & \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_t \end{bmatrix} & \longrightarrow \boxed{D} \longrightarrow & \begin{bmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \\ \vdots \\ \hat{\mu}_t \end{bmatrix}, \begin{bmatrix} \hat{\sigma}_1 \\ \hat{\sigma}_2 \\ \vdots \\ \hat{\sigma}_t \end{bmatrix} & \rightarrow & \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{bmatrix} \end{array}$$

NIM-VL

- NIM prototype!
- Captures complex stochastic input processes and simulates them
- Neural network of the form:

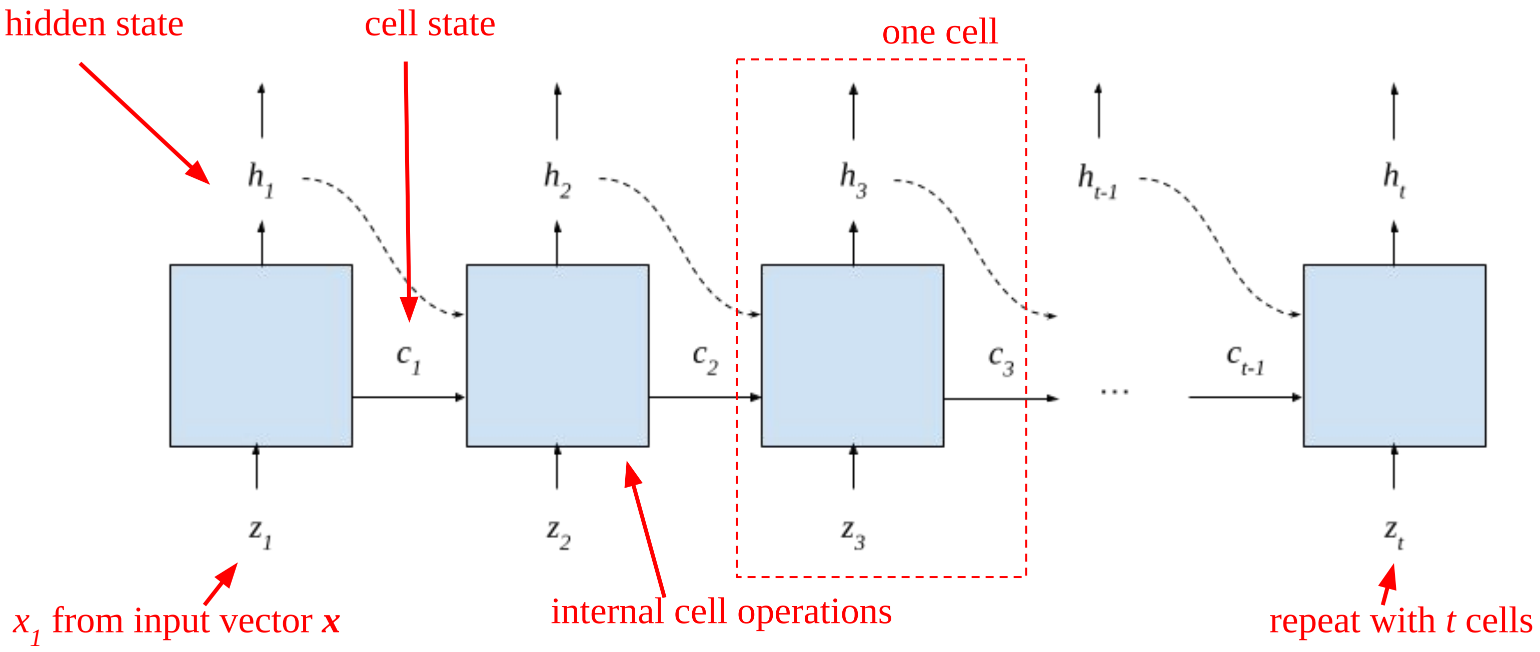
Generate data that will be distributed as a sample from the target distribution

Variational Autoencoder + LSTM layers

Long Short-Term Memory networks (LSTMs)

- Type of recurrent neural network (RNN)
- **Forms a memory of past inputs and concisely captures temporal patterns**
- Uses a series of repeating cells and hidden inputs
- Typically used as a single layer in a larger network

Long Short-Term Memory networks (LSTMs)



NIM-VL

- NIM prototype!
- Captures complex stochastic input processes and simulates them
- Neural network of the form:

Generate data that will be distributed as a sample from the target distribution

Form a memory of past inputs and concisely capture temporal patterns

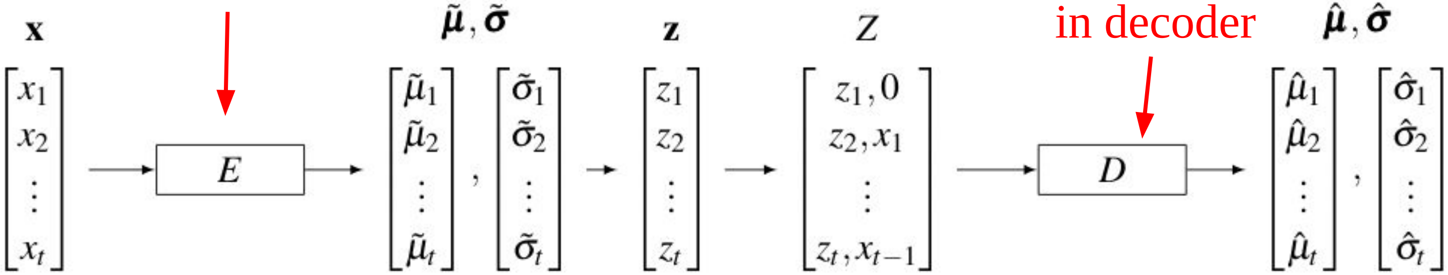
Variational Autoencoder + LSTM layers

NIM-VL at a glance

one LSTM layer in encoder

Assists the LSTM in the decoder

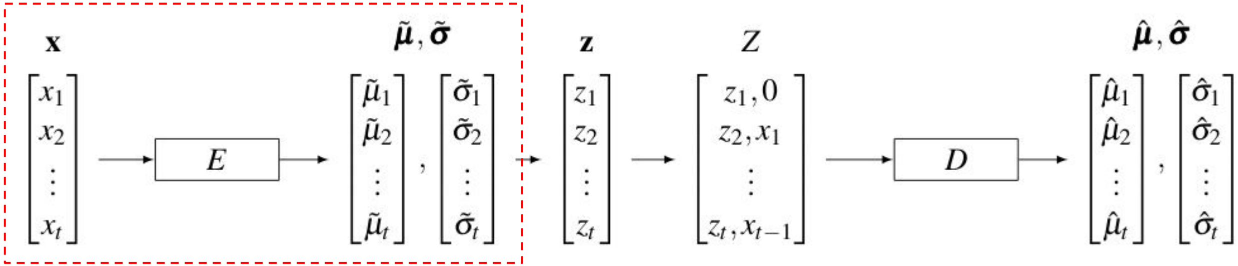
one LSTM layer in decoder



VAE architecture

Variational Autoencoder + LSTM layers

Training



1. Feed sample path $\mathbf{x}$ to the encoder E to get $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$.

Training

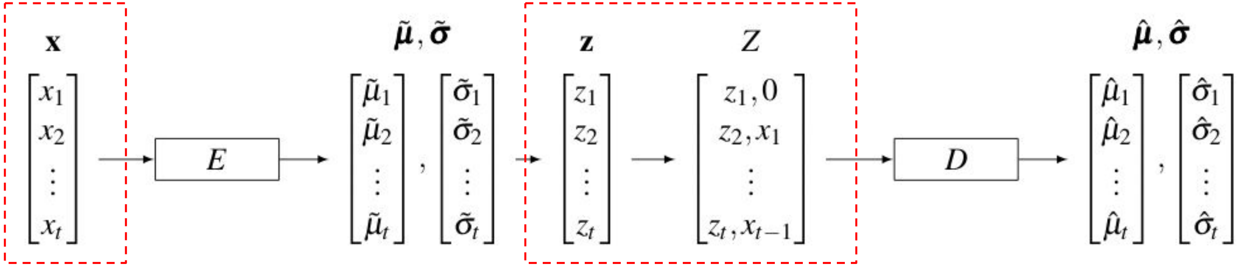
$$\begin{array}{ccccccccc}
\mathbf{x} & & & & \tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}} & & \mathbf{z} & & Z & & & & \hat{\boldsymbol{\mu}},\hat{\boldsymbol{\sigma}} \\
\begin{bmatrix}x_1\\x_2\\\vdots\\x_t\end{bmatrix} & \longrightarrow & \boxed{E} & \longrightarrow & \begin{bmatrix}\tilde{\mu}_1\\\tilde{\mu}_2\\\vdots\\\tilde{\mu}_t\end{bmatrix},\begin{bmatrix}\tilde{\sigma}_1\\\tilde{\sigma}_2\\\vdots\\\tilde{\sigma}_t\end{bmatrix} & \longrightarrow & \begin{bmatrix}z_1\\z_2\\\vdots\\z_t\end{bmatrix} & \longrightarrow & \begin{bmatrix}z_1,0\\z_2,x_1\\\vdots\\z_t,x_{t-1}\end{bmatrix} & \longrightarrow & \boxed{D} & \longrightarrow & \begin{bmatrix}\hat{\mu}_1\\\hat{\mu}_2\\\vdots\\\hat{\mu}_t\end{bmatrix},\begin{bmatrix}\hat{\sigma}_1\\\hat{\sigma}_2\\\vdots\\\hat{\sigma}_t\end{bmatrix}
\end{array}$$

1. Feed sample path $\mathbf{x}$ to the encoder E to get $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$.
2. Produce $\mathbf{z}$, by setting

$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i$$

for $i \in [1..t]$, where $\xi_i, \ldots, \xi_t$ are i.i.d. $N(0,1)$ random variables.

Training

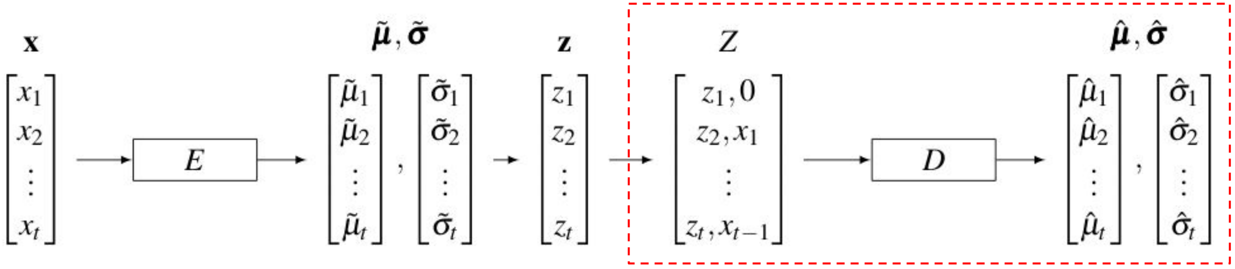


1. Feed sample path $\mathbf{x}$ to the encoder E to get $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$.
2. Produce $\mathbf{z}$, by setting

$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i$$

   for $i \in [1..t]$, where $\xi_i, \ldots, \xi_t$ are i.i.d. $N(0,1)$ random variables.
3. Create Z, where $Z_1 = [z_1, 0]$ and $Z_i = [z_i, x_{i-1}]$.

Training



1. Feed sample path $\mathbf{x}$ to the encoder E to get $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$.
2. Produce $\mathbf{z}$, by setting

$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i$$

   for $i \in [1..t]$, where $\xi_i, \ldots, \xi_t$ are i.i.d. $N(0,1)$ random variables.
3. Create Z, where $Z_1 = [z_1, 0]$ and $Z_i = [z_i, x_{i-1}]$.
4. Feed Z to the decoder D to get $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$.

Training

$$\begin{array}{c}\mathbf{x}\\ \begin{bmatrix}x_1\\x_2\\\vdots\\x_t\end{bmatrix}\end{array} \rightarrow \boxed{E} \rightarrow \begin{array}{c}\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}}\\ \begin{bmatrix}\tilde{\mu}_1\\\tilde{\mu}_2\\\vdots\\\tilde{\mu}_t\end{bmatrix},\begin{bmatrix}\tilde{\sigma}_1\\\tilde{\sigma}_2\\\vdots\\\tilde{\sigma}_t\end{bmatrix}\end{array} \rightarrow \begin{array}{c}\mathbf{z}\\ \begin{bmatrix}z_1\\z_2\\\vdots\\z_t\end{bmatrix}\end{array} \rightarrow \begin{array}{c}Z\\ \begin{bmatrix}z_1,0\\z_2,x_1\\\vdots\\z_t,x_{t-1}\end{bmatrix}\end{array} \rightarrow \boxed{D} \rightarrow \begin{array}{c}\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\sigma}}\\ \begin{bmatrix}\hat{\mu}_1\\\hat{\mu}_2\\\vdots\\\hat{\mu}_t\end{bmatrix},\begin{bmatrix}\hat{\sigma}_1\\\hat{\sigma}_2\\\vdots\\\hat{\sigma}_t\end{bmatrix}\end{array}$$

1. Feed sample path $\mathbf{x}$ to the encoder E to get $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$.
2. Produce $\mathbf{z}$, by setting

$$z_i = \tilde{\sigma}_i \xi_i + \tilde{\mu}_i$$

for $i \in [1..t]$, where $\xi_i, \ldots, \xi_t$ are i.i.d. $N(0,1)$ random variables.

3. Create Z, where $Z_1 = [z_1, 0]$ and $Z_i = [z_i, x_{i-1}]$.
4. Feed Z to the decoder D to get $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$.
5. Compute loss.

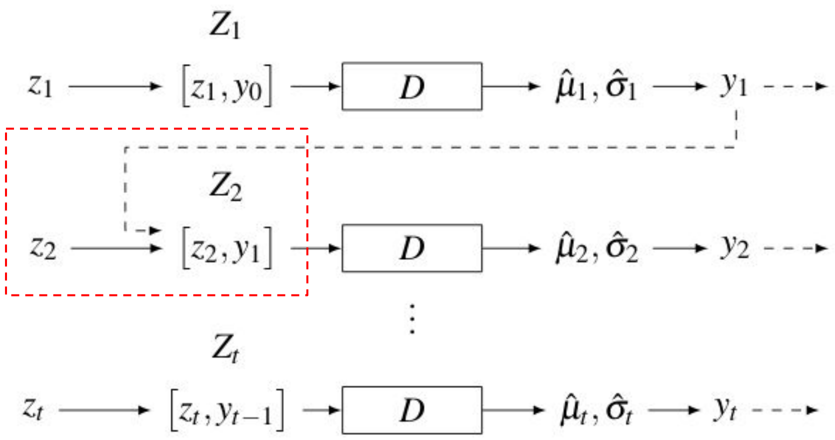
KL-Divergence
(regularization term)
minimize this

Log-Likelihood
(reconstruction term)
maximize this

$$L(\mathbf{x},\tilde{\boldsymbol{\mu}},\tilde{\boldsymbol{\sigma}},\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\sigma}}) = -\sum_{i=1}^{t}\left(\log\tilde{\sigma}_i^2 - \tilde{\mu}_i^2 - \tilde{\sigma}_i^2 + 1\right) + \sum_{i=1}^{t}\left(\log 2\pi + \log\hat{\sigma}_i^2 + \frac{(x_i - \hat{\mu}_i)^2}{\hat{\sigma}_i^2}\right)$$

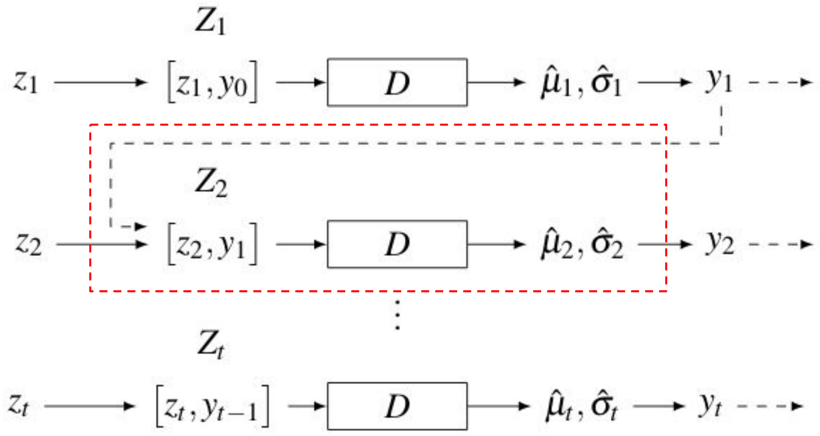
6. Backpropagate through the encoder and decoder. (basically gradient descent)

Generating sample paths



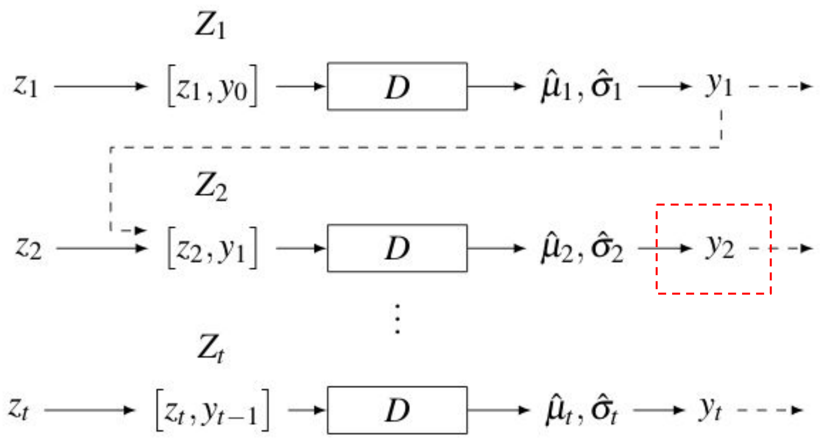
1. At iteration step i, draw variate $z_i \sim N(0,1)$ and combine it with y_{i-1} to create Z_i, where $y_0 = 0$.

Generating sample paths



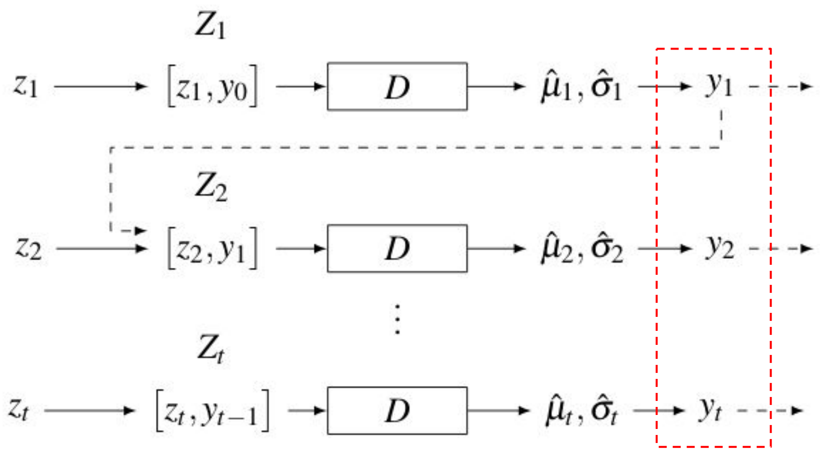
1. At iteration step i, draw variate $z_i \sim N(0,1)$ and combine it with y_{i-1} to create Z_i, where $y_0 = 0$.
2. Feed Z_i to the decoder D to produce $\hat{\mu}_i$ and $\hat{\sigma}_i$.

Generating sample paths



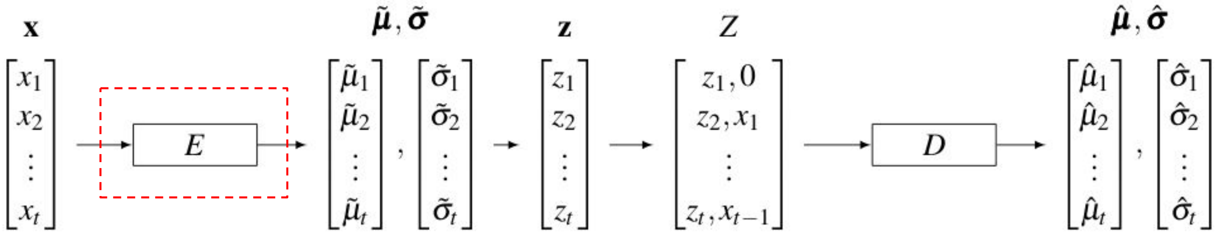
1. At iteration step i, draw variate $z_i \sim N(0,1)$ and combine it with y_{i-1} to create Z_i, where $y_0 = 0$.
2. Feed Z_i to the decoder D to produce $\hat{\mu}_i$ and $\hat{\sigma}_i$.
3. Compute $y_i = \hat{\sigma}_i w_i + \hat{\mu}_i$, where $w_i \sim N(0,1)$.

Generating sample paths

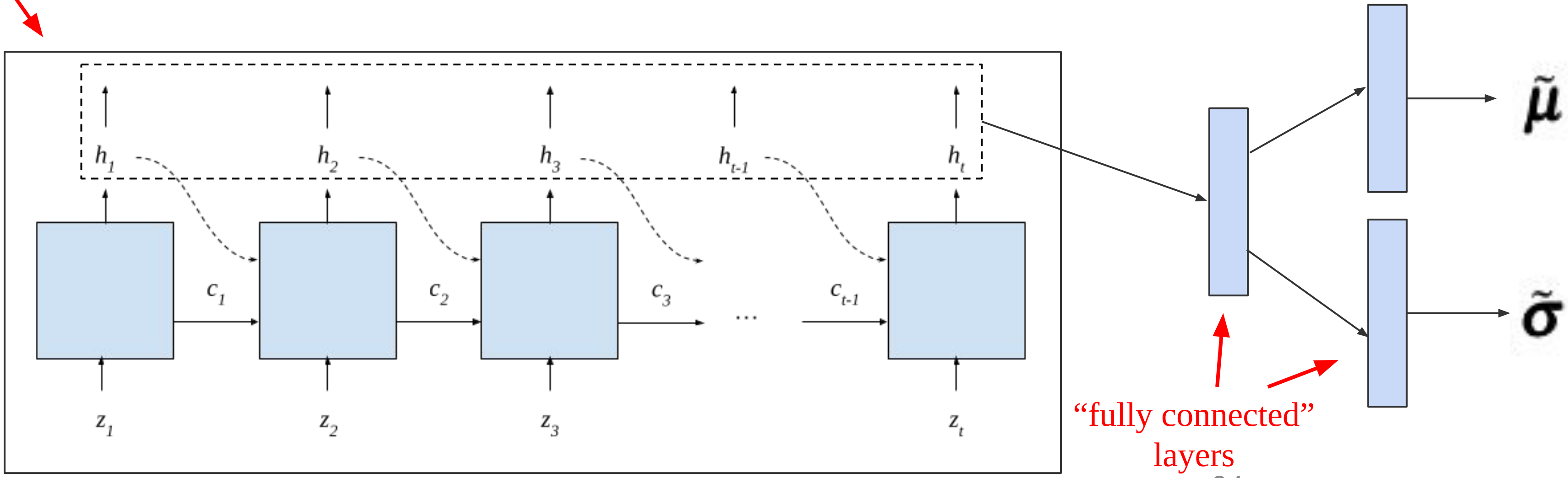


1. At iteration step i, draw variate $z_i \sim N(0,1)$ and combine it with y_{i-1} to create Z_i, where $y_0 = 0$.
2. Feed Z_i to the decoder D to produce $\hat{\mu}_i$ and $\hat{\sigma}_i$.
3. Compute $y_i = \hat{\sigma}_i w_i + \hat{\mu}_i$, where $w_i \sim N(0,1)$.
4. Repeat for iteration step $i+1$ until y_t is found.
5. Collect $y = (y_1, y_2, \ldots, y_t)$.

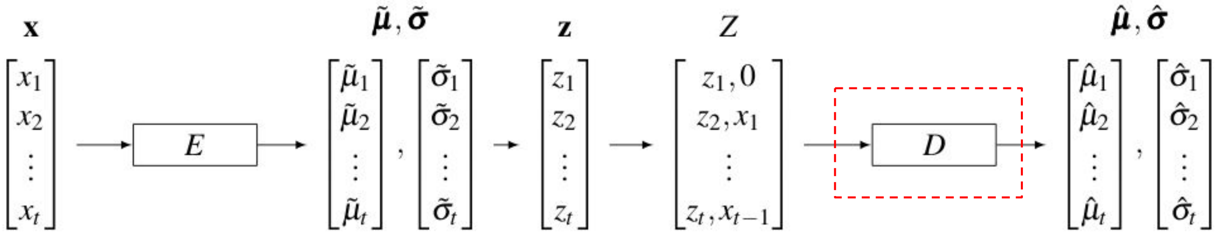
Encoder

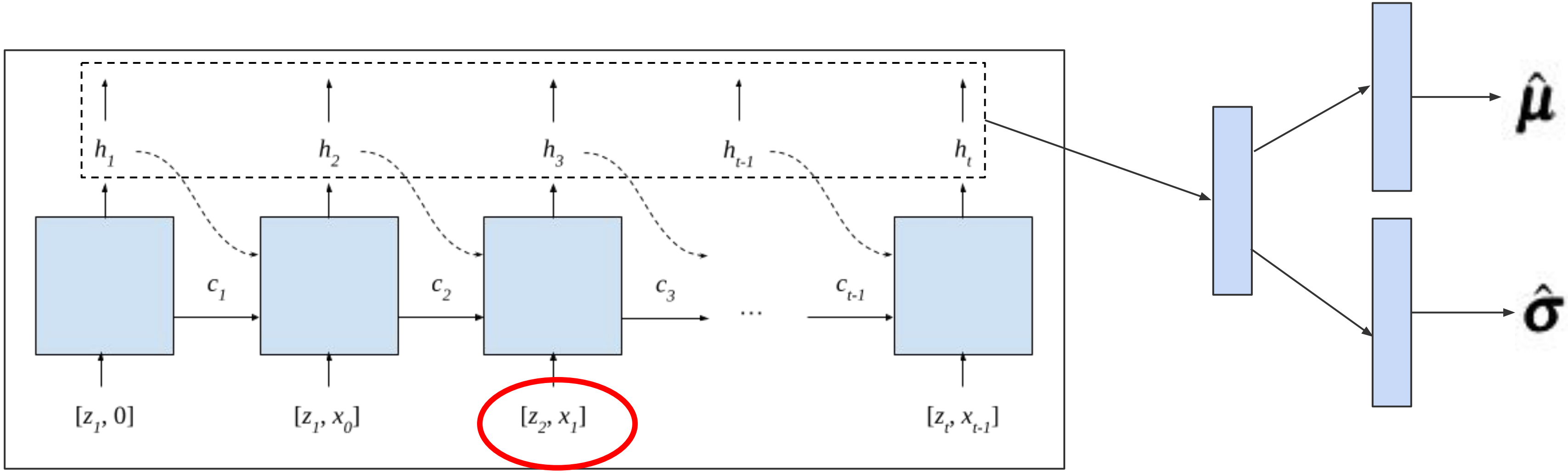


LSTM layer



Decoder





Ok, but how do I actually use it?

1. Train on n sample paths, each of length t
2. Generate m sample paths, each of length s
3. Use generated sample paths in simulation

Selected Experiments

- Goals:
  - Assess NIM-VL's ability to accurately capture complex input processes
  - Assess practicality of generation speed
- Experiments:
  - ARMA/ARMA mixture
  - NHPP
  - Queueing simulation

Nonstationary ARMA/ARMA mixture process

- Consider a mixture $\{X_i\}_{i\geq1}$ of two nonstationary ARMA(2,2) processes $\{A_i\}_{i\geq1}$ and $\{B_i\}_{i\geq1}$
- Standard Gaussian innovations
- Both processes in run parallel: at time *i*, we set $X_i=A_i$ with probability 0.5 and otherwise set $X_i=B_i$
- The parameters of the two processes are (0.95, −0.1; 0.2, 0.95) and (0.8, −0.3; 0.3, 0.7)

UMass Amherst | College of Information & Computer Sciences

Nonstationary ARMA/ARMA mixture process

- NIM-VL was trained on 10,000 ground truth sample paths, each of length 100
- NIM-VL is then used to generate 10,000 NIM-VL sample paths, each of length 100
- These NIM-VL sample paths are compared against 10,000 validation ground truth sample paths
(distinct from training data)

Nonstationary ARMA/ARMA mixture process

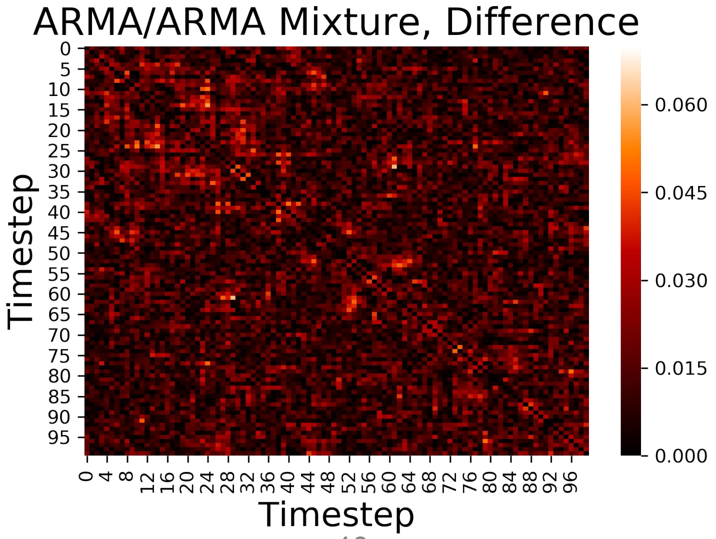
- Absolute difference in empirical correlation coefficients

- The largest absolute correlation difference value is 0.0609

$$\hat{\rho}_{ij}^{\mathrm{GT}} = \widehat{\mathrm{Corr}}[X_i, X_j] \text{ for } 1 \leq i, j \leq 100$$

$$\hat{\rho}_{ij}^{\mathrm{NIM}} = \widehat{\mathrm{Corr}}[X_i, X_j] \text{ for } 1 \leq i, j \leq 100$$

$$d_{ij} = |\hat{\rho}_{ij}^{\mathrm{NIM}} - \hat{\rho}_{ij}^{\mathrm{GT}}|$$



Nonhomogenous Poisson process (NHPP)

- Rate function $\lambda(t) = \frac{1}{2}\sin\left(\frac{\pi}{8}t\right) + \frac{3}{2}$
- Ground truth data is generated via thinning (Lewis and Shedler 1979)
- 10,000 sample paths, t = 100

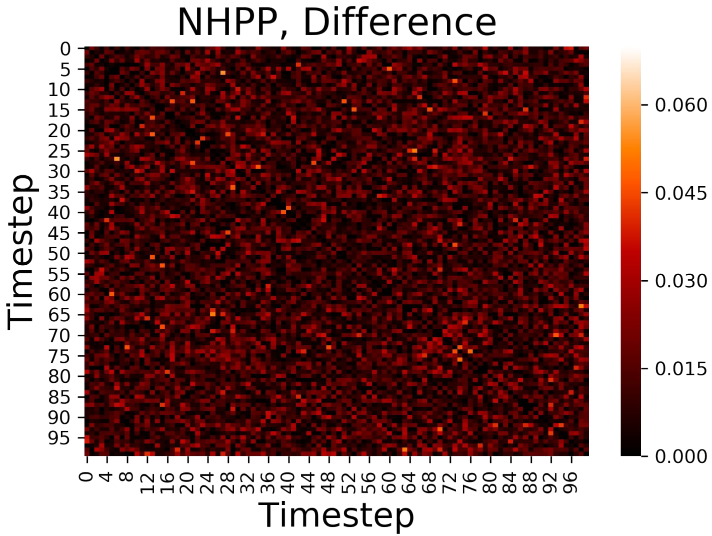
Nonhomogenous Poisson process (NHPP)

- The largest absolute correlation difference value is 0.0550

$$\hat{\rho}_{ij}^{\mathrm{GT}} = \widehat{\mathrm{Corr}}[X_i, X_j] \text{ for } 1 \leq i, j \leq 100$$

$$\hat{\rho}_{ij}^{\mathrm{NIM}} = \widehat{\mathrm{Corr}}[X_i, X_j] \text{ for } 1 \leq i, j \leq 100$$

$$d_{ij} = |\hat{\rho}_{ij}^{\mathrm{NIM}} - \hat{\rho}_{ij}^{\mathrm{GT}}|$$



Nonhomogenous Poisson process (NHPP)

- Empirical arrival rate function compared to ground truth $\lambda(t) = \frac{1}{2}\sin(\frac{\pi}{8}t) + \frac{3}{2}$

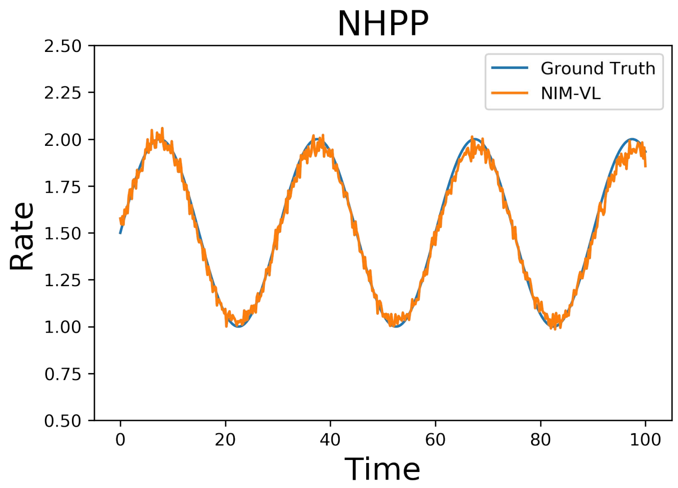
1. Compute the sequence of arrival times by taking partial sums of interarrival times

2. Divide the interval [0,100] into subintervals of length 0.2

3. Compute the average number of arrivals in each subinterval, where the average was taken over the 10,000 sample paths

Nonhomogenous Poisson process (NHPP)

- Empirical arrival rate function compared to ground truth $\lambda(t) = \frac{1}{2}\sin(\frac{\pi}{8}t) + \frac{3}{2}$
- Indicates good agreement

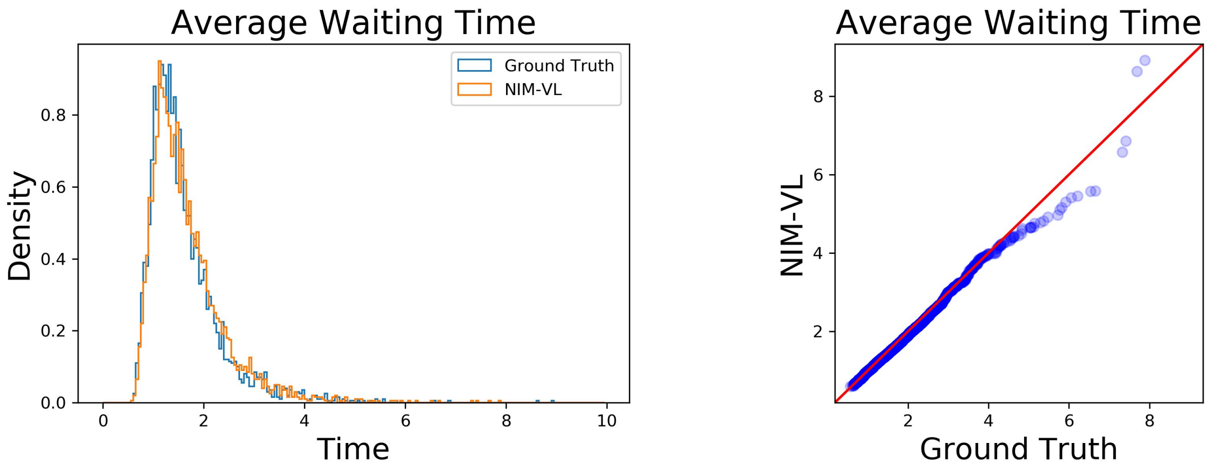


NHPP/Gamma/1 FIFO queue simulation

- Using NIM-VL to simulate the average waiting time $\overline{W}_{100}$ of the first 100 jobs in an NHPP/Gamma/1 FIFO queue
- Interarrival time process is NHPP $\lambda(t) = \frac{1}{2}\sin(\frac{\pi}{8}t) + \frac{3}{2}$
- Service times are i.i.d. Gamma (1.2, 0.4)
- Training with 1,000 sample paths with t = 50
  - (simulating t = 100)

NHPP/Gamma/1 FIFO queue simulation

- Empirical density of $\overline{W}_{100}$ over 4,000 simulation replications



Generation speed

- A trained NIM-VL model can generate 1,000 sequences of 1,000 learned NHPP interarrival times in roughly 0.85 seconds
  - PyTorch
  - Commodity 2019 MacBook Pro
- Training NIM-VL happens outside of simulation use, so training speed is not as crucial
- A priori knowledge about the distribution can be used to increase speed and accuracy
  - NIM-VM

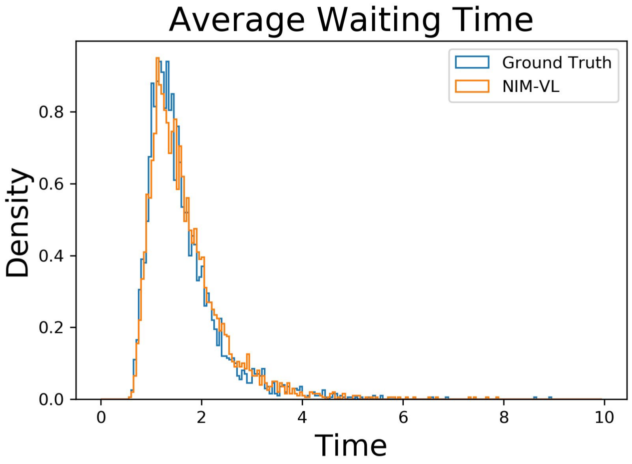
Conclusions and Future Work

- Conclusions:
  - Neural networks are a powerful tool, and could potentially lower the barrier for usage of simulation
  - NIM framework is promising for automatically capturing complex distributions and providing a means of sampling from those distributions
  - NIM-VLis able to capture some complex distributions

- Future work:
  - Formalizing theory
  - Metrics for accuracy evaluation
  - Alternate architectures (GANs, GRUs, etc)
  - Testing our approach in more complex simulations
  - Applications in privacy work

Questions?

Emily A. Herbert
emilyherbert@cs.umass.edu
people.cs.umass.edu/~emilyherbert



Goodness of fit statistic?

- Kolmogorov-Smirnov statistic
  - Multi-dimension statistic doesn’t capture complex patterns
- Anderson–Darling statistic
  - Multi-dimension statistic doesn’t capture complex patterns
- Bootstrap statistic
  - Unconventional
  - (Baringhaus ‘01)
- Log-Likelihood metric
  - Still thinking about this
  - How do you retrieve the density function?

Theory

- Currently thinking about this
- Universal approximation theorem:
  - Multilayer feedforward networks are capable of approximating any measurable function
  - (Hornik'91)
- For i.i.d. RV x with strictly increasing cdf F and a N(0,1) RV z, we have

$$g(z) = F^{-1}(\Phi(z)) \sim F$$

- Thinking inverse transform for time-dependent processes
- Thinking about combining inverse transform with universal approximation theorem